Journal of Software Engineering for Robotics 8(1), December 2017, 104-115
ISSN: 2035-3928

Consistent Geometric Estimation based on a World Model describing Logical Relationships and Sensor Interpretation

Andreas SCHIERL* Alwin HOFFMANN Wolfgang REIF
Instute for Software & Systems Engineering, Augsburg University, Augsburg, Germany
{schierl,hoffmann,reif}@isse.de

Abstract—When working with multiple independent mobile robots, each has a different knowledge about its environment, based on its available sensors. This paper proposes an approach that allows working with these different views by independently modeling the common logical relationships between the elements in the scene and the meaning of device-specific sensor data. Using these models, for each robot an estimation process can automatically be derived. This process combines and processes the available information to fill in the unknown geometric relationships between the elements and reacts to changes in the logical relationships. The paper presents two different ways of deriving the estimation process, one of which configures a standard unscented Kalman filter. The proposed approach facilitates the consistent coordination of the robots through an application that works on a common world model, while for execution each robot uses its available data and estimations. The approach is demonstrated in two case studies: one with two cooperating mobile robots tracked with an optical tracking system that hand over an object while passing each other, and one with a more challenging sensor constellation.

Index Terms—Software Architectures; World Modeling; Estimation; Kalman Filter; Mobile Robots; Robot Cooperation

1 INTRODUCTION

WHEN working with multiple independent mobile robots, each knows different aspects about its environment. These knowledge differences are based on the sensors available to the individual robots: While for typical industrial robots the positions of workpieces and tasks are exactly defined and ensured through fixtures, mobile robots often work in an environment where this strict structure is not present. In conjunction with the low precision of mobile robot locomotion, geometric uncertainty exists to an extent so that has to be handled. Therefore, sensors are added – either integrated into the robot, or mounted in the environment – to resolve the uncertainty based on measurements, and to consistently update the world model – the representation of the application's beliefs about its environment – accordingly. In popular approaches, the processing of sensor data is explicitly implemented or configured for the given scenario to form the robot's world model.

In contrast, this paper proposes to independently define the logical model (consisting of the logical relationships) and the measurement model (defining the meaning of sensor measurements in a geometric context). Based on these definitions and available sensor data, the proposed framework derives an estimation pipeline for the current situation. This estimation pipeline is created at run-time and provides consistent data for different robots that are controlled from the same application, each working with its available data.

Hence, the main contribution of this paper is a proposed separation of logical relationships and sensor interpretation which allows automatic derivation of an estimation pipeline. In doing so, the estimation process no longer has to be configured manually, but emerges from the given relationships and measurements. Additionally, the estimation pipeline can automatically adapt to changes in the environment. For example, when a robot grasps an object, sensor data no longer corrects the position of the object in the environment, but rather its position in the gripper.

As a further contribution, this approach allows consistent world models between cooperating mobile robots: all robots work on a common logical model, while for each robot its available sensors are used to derive information about geometric aspects it knows about. This offers better reusability, because common logical models, but also definitions of sensor interpretations can be used in other contexts, independent from the exact estimations they yield there.

Regular paper – *Manuscript received July 15, 2017; revised November 15, 2017. Digital Object Identifier: 10.6092/JOSER_2017_08_01_p104*

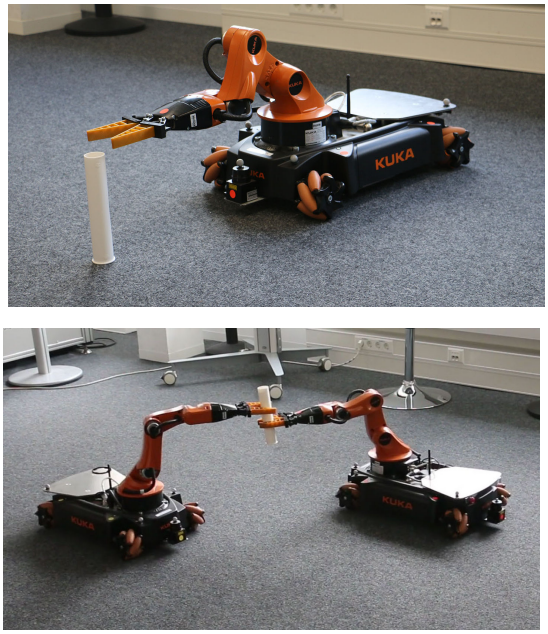


Fig. 1. Handing over an object between two moving youBots.

Extending the previous work [1], this paper gives more details about how to model logical relationships and observations, and introduces an algorithm to automatically derive a non-linear filter to handle situations with measurement noise or incomplete data.

As a real-world example, the interaction of two mobile robots is analyzed. One KUKA youBot picks up a baton placed in a predefined area of the room, and subsequently takes it to an area where it hands it over to a second youBot. This handover procedure takes place in motion while the youBots approach and pass each other. For this example, two youBots with laser scanners are used, along with a Vicon optical position tracking system. Both youBots are equipped with Vicon markers, while the baton is detected using the on-board laser scanners. Fig. 1 shows the two parts of the application, first picking up the baton and subsequently handing it over to the second youBot. The software for this example is implemented in an object-oriented fashion, representing the robots and baton as objects and correctly tracking their logical and geometric relationships during the process, so that the world model remains consistent with the real-world situation. In a second example, the advantages of using a Kalman filter variant are highlighted based on a situation where the robot position is estimated through multilateration.

The remainder of the paper is structured as follows: In Sect. 2, the overall approach is outlined. Then, the specification of logical relationships (Sect. 3.1) and sensor meaning (Sect. 3.2) is explained. Sect. 4 details how the estimation is performed based on these specifications. In Sect. 5, related approaches in theory as well as in existing robot frameworks

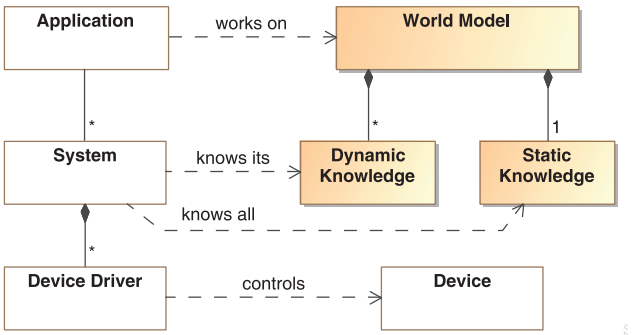


Fig. 2. Software structure for cooperating distributed robots, adopted from [2].

are compared. To conclude the paper, Sect. 6 explains the example implementation and experimental results and Sect. 7 gives a conclusion and outlook.

2 Approach

When considering cooperating robots, the underlying software architecture influences how and where data is available. Fig. 2 (adopted from [2]) shows a way typical software architectures for distributed cooperating robots can be structured. Each robotic device is represented and controlled by a device driver which is defined as the component that communicates – usually in a real-time capable way – with the device through a vendor-specific interface. It has to forward control inputs to the device and receive feedback from the device. One or multiple device drivers belong to a system where all knowledge is shared between the components (no longer necessarily with real-time guarantees, e. g. ROS nodes belonging to the same master). Hence, all components within one system are allowed to access each other's provided data, as well as to communicate with and send commands to each other.

To perform a desired task, an application controls the involved systems to coordinate the work flow (e. g., two systems in the example of cooperating but independent youBots). It reads data from controlled systems and sends commands to them in order to have the corresponding devices execute the overall task. Each application performs its task based on its knowledge about controlled devices, systems and the environment. This includes geometric information such as positions and orientations of the relevant objects, as well as information about the structure of (parts of) the environment (e. g., topologies, maps), physical data (e. g., mass, friction) or shape data (e. g., 3D models). The world model data can be differentiated into dynamic and static knowledge. While static knowledge (e. g., given maps, shapes or physical data) is valid and available everywhere, dynamic knowledge (e. g., positions or sensor data) may be known in only one system or be different among different systems. The latter can be the case for the position estimate of mobile robots.

The environment of a robot consists of various physical objects, each at its respective position. The contribution of the presented approach is to propose how to model logical relationships between these objects, i. e., the topology of the known environment, as static knowledge in order to infer dynamic knowledge automatically and in a consistent way. Inferring a consistent world model is especially important in the case of multiple systems in one application, such as distributed robots that have to work together to achieve a task. For example, the on-board sensors of a robot are only accessible in the system of the robot and, thus, can only be used for commands regarding this system. The responsibility for static knowledge, as it is the common part of the world model, lies at the application. Hence, the application manages topology changes, e. g., if and how a priori unknown objects are integrated into the topology.

Depending on the amount of structure in the environment, some geometric positions of objects are constant and can thus directly be modeled as static knowledge. Other positions however change over time and, thus, cannot be given exactly for an application that should be reusable later. Still, in order to work with them the application has to know about the existence of these objects and their logical relationships to further objects. Although these objects do have an exact position in the physical world, the application initially has no precise position information, which limits the amount of interaction that can be performed with these items.

To improve this situation, sensing can be employed to give the robot a glance of its environment. Based on sensor data, some positions (and velocities) of objects can be recovered, contributing to the geometry and, thus, dynamic knowledge of the application. To facilitate this, the interpretation of sensor measurements in a geometric context has to be defined. At application run-time, these interpretation definitions and incoming sensor data can be used to update the unknown or uncertain parts of the world model, so that it reflects a consistent interpretation of the received sensor data. In software, this process is performed through the introduction of logical and geometric relations, along with observations defining the semantics of sensor data and estimations automatically derived from this information.

Fig. 3 gives an overview of the concepts used to express this information. A robot's or to be precise a system's knowledge about itself and its surrounding world is called a *World View* and consists of a set of *Relation*s. A *Relation* correlates spatial features of objects like robots, workpieces or obstacles in the application. Such spatial features are modeled as *Frame*s, which represent Cartesian coordinate systems. One type of *Relation*s are *Logical Relation*s. Those describe statically known logical correlations among objects' *Frame*s. Objects that are connected to each other in a constant way, like a robot being mounted on a table, should be represented as *Static Connection*s. Other tight, but variable connections, like the relation between a robot joint and its associated links, should

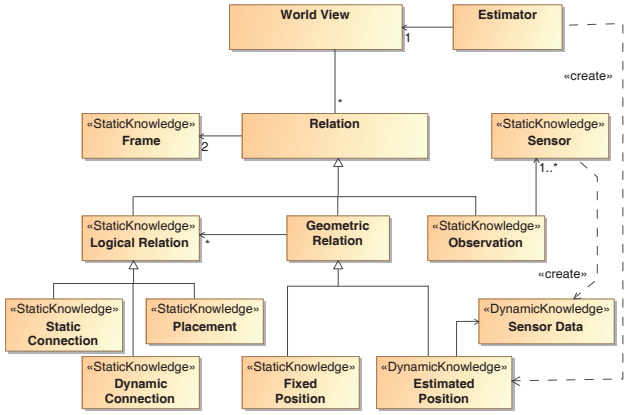


Fig. 3. Concept model for describing both the static and dynamic knowledge of a robotic application using different kinds of *Relations* between spatial features of physical objects.

be modeled as *Dynamic Connection*s. For rather loose and changing correlations like objects being placed somewhere on the ground in the vicinity of another object, *Placement*s are available.

The second type of *Relation*s, *Geometric Relation*s, form the geometric model and can describe the geometric position of an object relative to another, e. g., as a transformation matrix. *Geometric Relation*s correspond to (sequences of) *Logical Relation*s. They can either be constant (*Fixed Position*, e. g., for the position where the robot is mounted on the table), or be calculated from sensor values (*Estimated Position*).

However, *Estimated Position*s and their computation rule do not have to be defined explicitly, but may emerge automatically during application runtime. The application only has to define the interpretation of a sensor as *Observation*s. Because an *Observation* is a special *Relation*, it describes a correlation between two *Frame*s. Based on these *Observation*s and the *Logical Relation*s, so-called *Estimator*s automatically derive ways to process sensors in order to recover geometric information about unknown positions, and provide them as computation rules or other time-variable data for *Estimated Position*s computed from *Sensor Data*.

3 Defining the Logical Model and Sensor Interpretation

The basic ingredients of the world model are spatial features, called (coordinate) *Frame*s. Each *Frame* represents a (named) position in space, including an orientation. However note that a *Frame* per se does not know its absolute position in space – the position is only defined relative to other *Frame*s through *Geometric Relation*s. Concerning these spatial features, a logical structure exists that is modeled in the logical model. Additionally, some features are part or target

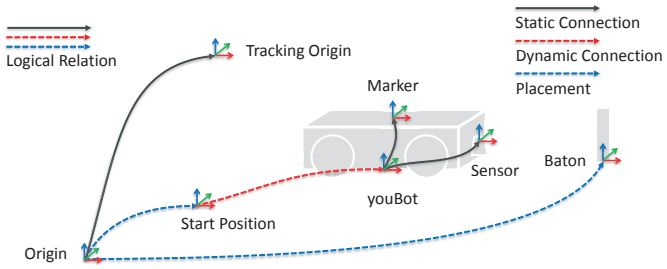


Fig. 4. Logical model of the youBot environment

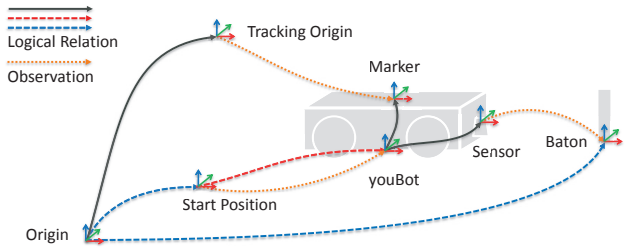


Fig. 5. Excerpt of the *World model* for a youBot platform tracked using an optical tracking system.

of sensor measurements which have to be interpreted in their corresponding context.

3.1 Defining the Logical Model

An application or robot model can establish *Relation*s between *Frame*s. As a basis of the environment model, *Logical Relation*s describe that certain *Frame*s are connected to each other, and include information about the durability of the link. The structure of *Frame*s and *Logical Relation*s forms an undirected graph, which allows navigation between different *Frame*s if a sequence of *Logical Relation*s exists that forms a path between the two *Frame*s.

Fig. 4 gives an example of *Frame*s and *Logical Relation*s. *Frame*s are depicted as named groups of arrows, which give the position as the intersection between the three arrows. *Logical relation*s are shown as arrows between the *Frame*s. The example shows *Frame*s for a *Start Position* and a *Tracking Origin* which are linked to an *Origin* frame, along with a *youBot* that is linked to its *Start Position*. Additionally, a *Marker* frame representing the position that can be tracked by the optical tracking system is connected to the *youBot*, as well as the *Sensor* frame where the laserscanner is mounted. Additionally, a *Baton* is placed on the ground and thus connected to the *Origin*.

In this scenario, the different relationships have a different meaning, and are thus represented as different types of relationships. The position of the *Tracking Origin* relative to the *Origin* is fixed and given, and thus represented as a *Static Connection*. Similarly, the position of the *Marker* and *Sensor* relative to the *youBot* is fixed and constant. In contrast, the *Start Position* of the youBot in the world (relative to *Origin*) is not fixed, but the youBot is rather placed into the world. Thus, this relationship is a *Placement*, as well as the one to the *Baton*. The position of the *youBot* relative to its *Start Position* is controlled by the youBot and is thus modeled as a *Dynamic Connection*. Consequently, *Static Connection* and *Dynamic Connection* represent persistent relations that will exist for the entire run time of an application, while *Placement*s are transient and can be removed. *Placement*s and *Static Connection*s are assumed to model a relationship with constant transformation, while the transformation of *Dynamic Connection*s varies over time.

Additionally, the *Dynamic Connection*s give a model of possible motions. This model consists of state variables for position and velocity, as well as a mapping that – based on the current value of the state variables – computes the Cartesian transformation and twist between the corresponding frames. In the case of the youBot platform, the connection has three position state variables (for the X and Y translation as well as the Z rotation) and three velocity state variables (for forward and lateral motion as well as rotation). In contrast, the youBot arm contains five connections with one state variable for position and velocity each (describing the rotation of the corresponding joint).

When the baton is grasped by the youBot, the *Placement* from *Origin* to *Baton* is removed, and a new Placement from the *Gripper* to the *Baton* is established. This topology change that occurs at application run time has an influence on the geometric relations and is detailed in the following sections.

3.2 Defining the Sensor Interpretation

To derive a geometric model elaborating the logical model defined above, *Geometric Relation*s providing transformations and velocities have to be added. For relations that are not given statically, this transformation often has to be derived from sensor measurements. To achieve this, the semantics of measured sensor data has to be defined in the form of *Observations* that describe that a certain aspect of the *World model* is measured by a given sensor, or can be calculated from given *Sensor Data*.

Looking into robot environments, different types of sensors and correspondingly observations occur: Most logical relations inside *Actuator*s can be measured through proprioceptive sensors. There, the measurements typically follow the logical structure of the object: In the case of the youBot base, four observations define that the sensor data provided by the wheel encoders describe the positions of the wheels relative

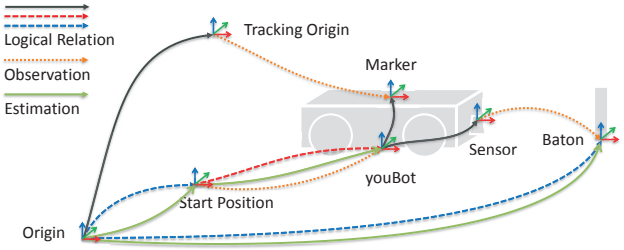


Fig. 6. Estimations established for a youBot platform when all sensors are available

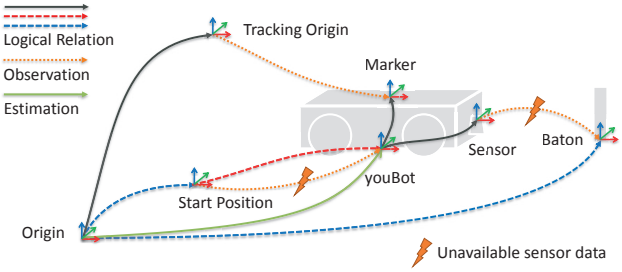


Fig. 7. Estimations established for a youBot platform from an outside view if only tracking data is available

to their wheel mount frames. Similarly, odometry calculations based on wheel encoders provide the transformation between *Start Position* and the *youBot*. These observations describe transformations that directly follow a logical relation.

For other logical relations, however, this is not the case: Assuming the youBot platform is driving on the floor starting at a position not exactly known, application geometry is usually modeled as a *Placement* from the *Origin* frame to the youBot *Start Position* (cf. Fig. 4). Its transformation however cannot be measured directly, because the *Start Position* is an intermediate concept that does not have a direct representation in the physical world (at least after the youBot platform has moved). Fig. 5 as an extension to Fig. 4 gives an overview of this situation. It shows a youBot platform on the floor that is tracked through an optical tracking system. In addition to the logical model, three full pose observations are given (represented as dotted arrows). As mentioned above, the position of the *youBot* relative to its *Start Position* is given by odometry sensors. Additionally, the *Marker* of the youBot is tracked using the tracking system, so the observation between *Tracking Origin* and *Marker* is given by the tracking system. Furthermore, the *Baton* is tracked from the *Sensor* through the laserscanner (with some post-processing). The transformations defined by the second and third observation cannot directly be used to describe the transformation of a *Logical Relation*, but first have to be converted.

Looking at the three observations given here, the first and third are only applicable on the youBot system, because they use sensor data that is only available to the youBot. The second observation however can be used wherever the tracking system's data is available, e.g. for a second cooperating youBot.

These observations share the commonality that they observe the full pose, i.e. they provide information about the position as well as orientation of the frame. In contrast, other observations may only observe a certain aspect about the transformation, e.g. the distance, and are described in Sect. 4.3.

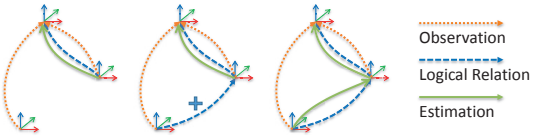
4 Using Estimation to Derive Geometric Information

To coordinate the interplay between logical relations and observations and to provide estimations for the uncertainties present, the concept of *Estimator*s on the system level is introduced. An *Estimator* listens to changes in the logical relations as well as observations, and augments its *World View* with *Estimated Position*s that reflect one *Logical Relation* or shortcut multiple ones. These *Estimated Position*s can be seen as estimation pipelines processing the data provided by sensors as defined in the observations, and give computation rules for how to calculate the transformation and velocity if the sensor's dynamic knowledge is available in the corresponding system.

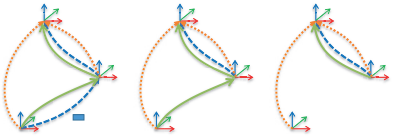
Given the example in Fig. 6, the *Estimator* creates three *Estimated Position*s: The first relation goes from the *Start Position* to the *youBot* frame and takes the value from the odometry sensor as a transformation. The second relation from *Origin* to *Start Position* is a bit more complex: the transformation can be combined from the transformation from *Origin* to *Tracking Origin*, followed by the transformation provided by the *Observation*, the transformation from the *Marker* to the *youBot* and the transformation from *youBot* to *Start Position*. For the last part, the transformation provided by the first estimated *Relation* has to be used. Similarly, the third transformation can be computed using the observed position of the *Baton*, along with the estimated position of the youBot. However, when another robot observes the youBot from outside, the sensor data for the odometry *Observation* as well as the laserscanner measurements are not available. In this case, the *Estimator* has to create an *Estimated Position* directly from the *Origin* frame to *youBot* (cf. Fig. 7).

4.1 Establishing and Updating the Geometric Model

At application run-time, *Estimator* implementations work as listeners that react to changes in the *Observation*s, *Logical* and *Geometric Relation*s. The *Estimator* tries to find cycles in the graph formed by *Logical Relation*s and *Observation*s, and uses their information to build new known *Estimated Position*s.



(a) Adding a Logical Relation



(b) Removing a Logical Relation

Fig. 8. Estimation modifications when changing Logical Relations

When searching for cycles, it tries to minimize the number of *Logical Relation*s without corresponding *Geometric Relations* required to form estimations in order to keep the estimations structurally as close to the logical structure as possible.

Algorithm 1 gives an overview about this process. In cycles in the relation graph, two situations can occur: if there is a loop in the graph of *Logical Relation*s, and at least one *Geometric Relation* describing a part of the circle, the unknown geometric transformation of the *Logical Relation*s in the loop can be estimated. For example, when grasping the baton placed on the ground, the baton initially has a *Placement* (with *Estimated Position*) on the ground, while a new *Placement* is added connecting it to the gripper. This new *Placement* can be estimated, because a complete path of *Geometric Relation*s exists that describes the transformation of the baton relative to the gripper. In another situation, a cycle for the *Logical Relation* may include an *Observation*. Then, the cycle consists of an *Observation*, a (maybe empty) sequence of *Geometric Relation*s, followed by a sequence of *Logical Relation*s (that will be estimated and should thus be as short as possible), and a (possibly empty) sequence of *Geometric Relation*s closing the loop. In both types of cycles, there is a sequence of logical relations ($lseq$) as well as a sequence of geometric relations and observations ($gseq$) that both describe a path between the same frames, while $gseq$ contains at least one relation that is not an estimated position (lines 6 – 10).

For a found cycle, the *Estimator* takes the *Observation*'s transformation and velocity and converts it for the *Frame*s forming the start and end of the *Logical Relation* sequence, so that calculation rules describing the overall behavior (position and velocity) of the *Logical Relation* sequence are available (lines 13 – 26). These calculations are then used to define the *Estimated Position*, and subsequently evaluated whenever this aspect of the *World model* is accessed by the application, e.g. during motion planning (lines 27 – 29).

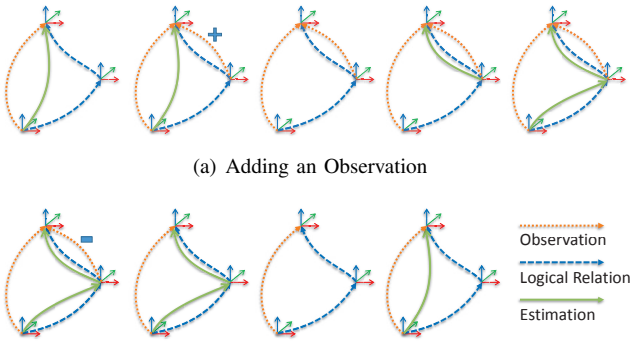
When a *Logical Relation* is added (cf. Fig. 8(a)), an *Estimator* checks if a sequence of *Geometric Relation*s exists

Algorithm 1 Basic Estimation Process

1: Using R_L ▷ Logical Relations
2: Using R_G ▷ Geometric Relations
3: Using O ▷ Observations
4: $R_E \leftarrow \emptyset$ ▷ Created Estimations
5: **repeat**
6: $lseq, gseq \leftarrow$ shortest sequence $lseq$ with
7: $lseq \subset R_L \ \wedge \ gseq \subset R_G \cup O \ \wedge$ ▷ Find cycles
8: $lseq.from = gseq.from \ \wedge$ ▷ linking logical and
9: $lseq.to = gseq.to \ \wedge$ ▷ geometric model
10: $gseq \setminus R_E \neq \emptyset$ ▷ that are non-trivial
11: $from \leftarrow lseq.from$
12: $to \leftarrow lseq.to$
13: $trans \leftarrow$IdentityMatrix() ▷ Collect geometric
14: $vel \leftarrow$ZeroTwist() ▷ transformation and twist
15: $cur \leftarrow from$
16: **for all** $g \in gseq$ **do** ▷ Follow the geometric sequence
17: **if** $g.from = cur$ **then**
18: $trans \leftarrow trans \cdot g.trans$
19: $vel \leftarrow$ CombineTwist$(vel, g.vel)$
20: $cur \leftarrow g.to$
21: **else** ▷ and also handle reversed relations
22: $trans \leftarrow trans \cdot g.trans^{-1}$
23: $vel \leftarrow$ CombineTwist$(vel, g.vel^{-1})$
24: $cur \leftarrow g.from$
25: **end if**
26: **end for**
27: $e \leftarrow$ new EstimatedPosition$(from, to, trans, vel)$
28: $R_E \leftarrow R_E \cup \{e\}$ ▷ Remember new estimation and
29: $R_G \leftarrow R_G \cup \{e\}$ ▷ add as geometric relation
30: **until** no sequence $lseq$ is found

between the corresponding frames, and if so adds a corresponding *Estimated Position*. Otherwise, if any of the known *Observation*s can be used to form a cycle in the *World view* including the new *Logical Relation*, an *Estimated Position* is established based on the *Logical Relation* and *Observation*. For *Geometric Relation*s added, the *Estimator* checks if the new *Geometric Relation* has an effect on any of the existing estimations. This is the case when the *Relation* influences the first or last *Logical Relation* resolved through the estimation, causing the *Estimated Position* to be recreated based on the new situation. When a *Logical Relation* is removed (cf. Fig. 8(b)), the corresponding estimations become invalid and are removed. Similarly, removing *Geometric Relation*s can invalidate estimations if they occurred in the cycle used to build the estimation, so new estimations for the corresponding logical connections have to be built.

Adding an *Observation* (cf. Fig. 9(a)) may allow resolving one of the *Logical Relation*s by forming a cycle including the new *Observation* and building a corresponding estimation. If the resulting estimation contains a subset of the *Logical*



(a) Adding an Observation

(b) Removing an Observation

Fig. 9. Estimation modifications when changing Observations

*Relation*s used in another estimation, the latter estimation is removed and recreated with the option to use the newly built estimation, bringing the estimations closer to the logical structure (e. g. in the last step of Fig. 9(a)). When an *Observation* is removed that has been used by estimations (cf. Fig. 9(b)), the corresponding *Estimation*s are removed and new cycles for the corresponding *Logical Relation*s are searched.

As an example, a workpiece tracked by a sensor is considered: The workpiece lying on the floor is connected to the ground using a *Placement*. Additionally, an *Observation* is given that provides the position of the workpiece relative to the sensor, and thus allows the calculation of a transformation for the *Placement*. However, once the workpiece is grasped, the *Placement* on the ground is removed and another *Placement* to the gripper is added. In this situation, the same *Observation* now has to be used to provide another *Estimated Position*. This case is automatically handled by *Estimator*s.

4.2 Handling Delayed and Sporadic Data

In simple cases, the *Estimator* can assume that all sensors used in *Observation*s are precise and provide values all the time, without any time delay. Then, the estimator can use the latest position data from all Observations to define its *Estimated Position*s.

For *Observation*s referencing *Sensor Data* that are only provided sporadically or delayed, a more complex *Estimator* is required, however still following the method outlined in Sect. 4.1. It still assumes that all *Observation*s are precise, but accepts that some *Observation*s are only provided infrequently, but with correct time stamps (e.g. when a tracking system temporarily loses an object or a lag in wireless network communication occurs). It further respects the types of *Logical Relation*s. For *Placement*s and *Static Connection*s, it assumes that they are actually constant and only have to be changed to correct measurement errors (which is true for objects placed on the ground, as well as for the *Start Position* of a youBot). In contrast, for Dynamic Connections extrapolating

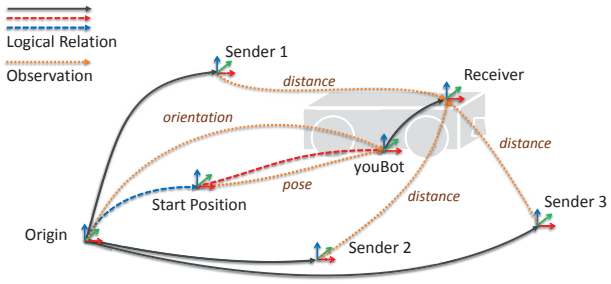


Fig. 10. Excerpt of the *World model* for a youBot platform tracked using tree distance measurements.

with constant velocity is an appropriate estimation. In this case, *Estimated Position*s are created between the same *Frame*s as in the simple case, however, transformations are taken from a consistent time snapshot and extrapolated if required. While supporting more use cases, this estimator has the disadvantage of increased memory usage and computation time: to calculate estimations for a consistent time, it has to keep track of previous values and times of the sensor data, and has to select a corresponding time to use the data from. Especially on resource-constrained systems that need an estimator that runs at a high frequency with hard real-time guarantees, using a simple estimator instead can thus be a better option when no greater time delays are to be expected for the sensor data.

4.3 Handling Partial and Noisy Sensor Data

Apart from situations where *Observation*s provide the exact position of a *Frame*, working in real-world noisy environments suggest the use of extensions towards more complex observations and estimations. However, these require more complex estimation processing than those described in Sect. 4.1.

The first extension is to support *Observation*s that do not give the full transformation between two frames: One sensor might measure the distance to a given landmark, yielding an *Observation* that describes the distance between two *Frame*s. Another sensor might be able to determine the height of a quadcopter, e.g. through ultrasound or barometric pressure. Here an *Observation* only provides the Z coordinate of a transformation. Furthermore, many sensors exhibit sensor noise that is too big to be neglected. This fact has to be handled, e.g. by modeling the *Sensor Data* belonging to an *Observation* as distorted by Gaussian noise with a given covariance.

As an example, Fig. 10 shows the situation when the youBot is not tracked using an optical tracking system, but rather has a sensor (*Receiver*) that provide the distance to given landmarks (*Sender 1*, *Sender 2* and *Sender 3*). To model this, three distance observations are used between the landmarks and the robot, defining that the values provided by the sensor are to be interpreted as the distance to the corresponding frames. Additionally, the youBot is assumed to have a magnetometer,

so that it can measure its orientation compared to the magnetic north pole, which is modeled as an orientation observation between *Origin* (which is assumed to be aligned with magnetic north) and the *youBot*. Finally, the position of the *youBot* relative to its *Start Position* is given through odometry sensors. Using this model, the unknown logical relation between *Origin* and the *Start Position* can be estimated, as evaluated in Sect. 6.

Additionally, the model of possible motions given by *Dynamic Connection*s can be used to improve estimations: through the explicit definition of degrees of freedom and state variables, it is not required to estimate the full 6 degrees of freedom as done in Sect. 4.1, but rather to limit the estimation to the aspects that can really change. Additionally, the dynamic behavior of the connections can better be estimated by using the velocity state variables of the *Dynamic Connection*s or the modeled relationship between different state variables. This allows to model one object with angular and linear velocity (e.g. a spaceship) as moving on in a straight line while spinning, while another (e.g. a car) moves on in a circles.

To make use of these extended models of observations and connections, an estimation scheme more complex than in Sect. 4.2 is required, however standard estimation techniques can be used. Here, our implementation uses an *Unscented Kalman Filter* [3] that is automatically created and configured from the existing logical relations and observations, as outlined in algorithm 2.

First, the state variables required for the system model are collected by examining all *Logical Relation*s for which no *Geometric Relation* exists. For *Dynamic Connection*s with defined state variables, the corresponding position and velocity state variables are included in the system model (lines 11 – 18). For other *Logical Relation*s (such as *Placement*s), state variables describing the unconstrained position and orientation are used. To further define the system model, a state transition model describing the derivatives of the position state variables is required. For *Dynamic Connection*s, this derivative is given by the relationship between modeled velocities and velocity state variables, whereas for other *Logical Relation*s the derivative is assumed to be zero (line 19).

Based on these state variables, a view v of the system is created that describes the resulting geometric state for given values of the state variables. This view takes the current geometric model (lines 5 – 7) and augments it with the geometric aspects given by the *Dynamic Connection*s for values of the used state variables (lines 21 – 24). Based on this view, all *Observation*s then provide their expected values for the given situation to be used as the filter's observation model (line 30 – 31). Additionally, the *Observation*s provide access to the real sensor values for the corresponding aspect, as well as their covariance (line 33).

These models describing the state transition and observation as functions of the state variables are used to instantiate a real-time component implementing an Unscented Kalman Filter, which is started and continuously fed with values for the sensors described by the *Observation*s (lines 35 – 37). This filter then provides an estimation for the state variables, which is in turn used to augment the world model with *Estimated Position*s according to the rules given in the corresponding *Dynamic Connection*s (lines 38 – 44).

Algorithm 2 Estimation using a Kalman Filter

1: Using R_L ▷ Logical Relations
2: Using R_G ▷ Geometric Relations
3: Using O ▷ Observations
4: $v \leftarrow \emptyset$
5: **for all** $r \in R_G$ **do** ▷ System view starts with
6: $v \leftarrow v \cup \{r \mapsto r.trans\}$ ▷ current situation
7: **end for**
8: $svs \leftarrow \emptyset$ ▷ Collect state variables,
9: $stm \leftarrow \emptyset$ ▷ state transition model
10: $est \leftarrow \emptyset$ ▷ and relations to estimate
11: **for all** $r \in R_L$ **do**
12: **if** $\nexists g \in R_G : r.from = g.from \wedge r.to = g.to$ **then**
13: ▷ For purely logical relations
14: **for all** $sv \in r.stateVariables$ **do**
15: ▷ create variables for states and
16: ▷ store their transition model
17: $var \leftarrow$new Variable()
18: $svs \leftarrow svs \cup \{sv \mapsto var\}$
19: $stm \leftarrow stm \cup \{sv \mapsto \text{transitionModel}(r, sv)\}$
20: **end for**
21: ▷ Store transformation in system view
22: $v \leftarrow v \cup \{r \mapsto \text{transformationModel}(r, svs)\}$
23: ▷ and remember to estimate r
24: $est \leftarrow est \cup r$
25: **end if**
26: **end for**
27: $om \leftarrow \emptyset$ ▷ Collect observation model and
28: $ov \leftarrow \emptyset$ ▷ sensor values for observation
29: **for all** $r \in O$ **do**
30: ▷ Observation model is based on system view
31: $om \leftarrow om \cup \text{observationModel}(o, v)$
32: ▷ while sensor values are concrete values
33: $ov \leftarrow ov \cup \text{sensorValue}(o)$
34: **end for**
35: ▷ Set up kalman filter for state variables, state transition
36: ▷ model, observation model and observation values
37: $f \leftarrow$ new KalmanFilter(svs, stm, om, ov)
38: **for all** $r \in est$ **do** ▷ Create estimated position
39: ▷ based on transformation model
40: $trans \leftarrow \text{transformationModel}(r, f.state)\}$
41: $e \leftarrow$ new EstimatedPosition$(r.from, r.to, trans)$
42: ▷ remember new estimation
43: $R_G \leftarrow R_G \cup \{e\}$
44: **end for**

Using this method, all part of the world model are modeled by a single Kalman filter, which increases the computational

complexity of the problem through a big number of state and observation variables. To improve this situation, unrelated parts of the world model can be split into different filters. To achieve this, a partitioning of *Logical Relation*s and *Observation*s has to be found so that all elements of each circle existing in the world model are in the same partition. Thus, for each variable in the observation model the used state variables (and their dependencies) are grouped into one partition each, and then all non-disjoint partitions are combined. For each resulting partition, a Kalman filter can be created that uses the corresponding observations and state transition model.

Splitting the model into different parts also improves the reaction to changes in the logical model: for all unchanged parts, the filter can continue to run, while for changed parts a new filter has to be initialized. The new filter should however still continue with the previous state and covariance estimation for all known state variables, and only re-initialize for new variables.

Even when using this (more complex) filtering scheme, the separation of models as proposed in this work allows keeping the logical and static geometric relations independent from the concrete estimation process and reusing the logical model for future applications.

5 Related Work

Looking at control theory, the process of integrating sensor data into the world model is similar to the concept of state observers used along with the state-space representation of a system model: a system model consists of a state equation describing the behavior of the system under the influence of the given system inputs, and an output equation that defines the relationship between system state and the measured variables. Standard estimation methods such as the *Kalman Filter* [4] for linear problems allow processing system inputs and measurements to recover the state of the system, tracking the uncertainty of the present state variables. For non-linear problems (that occur in robotics, e.g. through rotations that have an non-linear effect on the robot position when the robot drives forward), extensions of the *Kalman Filter* [5] are available such as the *Extended Kalman Filter* linearizing the problem around the given state, or the *Unscented Kalman Filter* that samples chosen points in the probability distribution. Furthermore, *Particle Filters* [6] allow tackling problems with greater uncertainty, such as an initial localization of a robot in a known map.

Looking at *ROS*, the frame graph is typically managed through the *tf* library [7], where a tree of geometric relationships between frames can be established. However, no semantic information is contained there, apart from a separation between static and dynamic transformations (through the */tf_static* and */tf* topics). Estimation and sensor integration for robot positions is typically performed through specialized components. According to ROS Enhancement Proposal 105 [8], mobile platforms should be modeled with three distinguished frames: the frame *map* represents the origin of the robot environment (and a corresponding map), while *odom* represents the starting point of the robot and *base* denotes the robot itself. The transformation between *map* and *base* does not have to be continuous, but may not drift over time, while the transformation between *odom* and *base* has to be continuous, but may exhibit unbounded drift. In the frame tree used in *ROS*, *odom* is the parent frame of *base*, while *map* is the parent of *odom*. This modeling is similar to the model used in the proposed approach, with *odom* representing the *Start Position* and *map* representing a fixed frame (such as the *Tracking Origin* or *Origin*). To handle sensor data, *robot_pose_ekf* [9] and later *robot_localization* as described by Moore et al. [10] provide different estimation algorithms as *ROS Node*s, including an *Extended* and *Unscented Kalman Filter*. These *Node*s can process various sensor data, including global positioning systems (GPS), inertial measurement units (IMU) and odometry (ODOM) data, and publish the transformation between *map* and *odom* or the transformation between *odom* and *base*. However, these estimation nodes have to be configured manually, and cannot be used with topology changes: for an object first tracked using multiple sensors, and then grasped by a robot, the parent frame changes, and thus the estimation node has to be reconfigured.

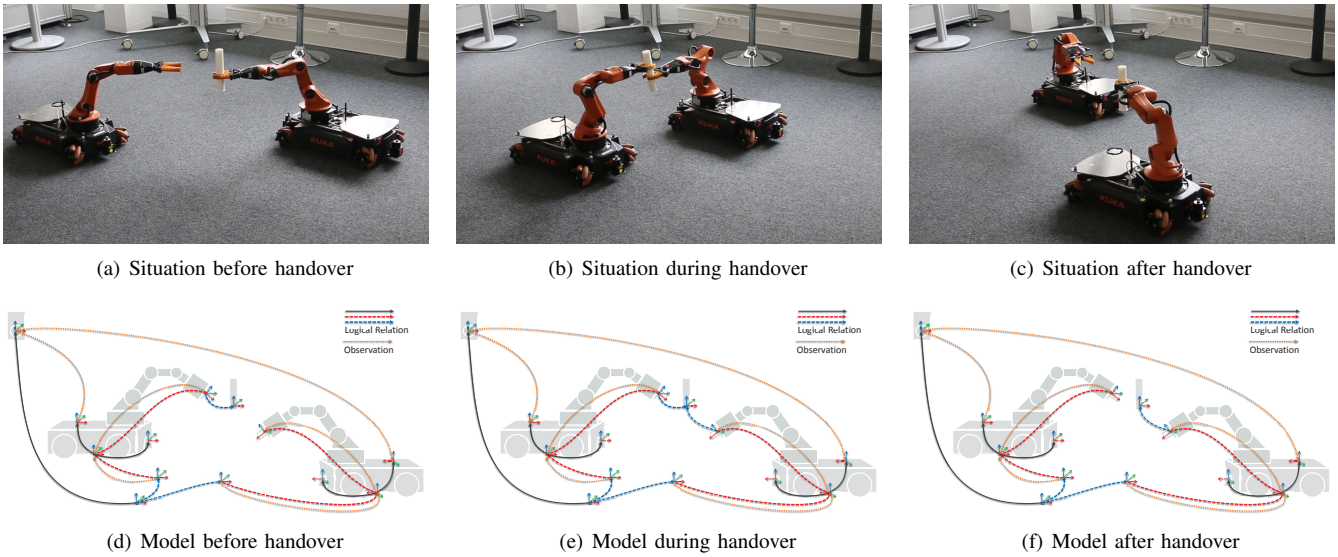
In *OROCOS*, sensor data and estimation can e.g. be handled through the *iTaSC* framework [11]. There, uncertainty can be defined for object or feature coordinates, which is then processed during task execution using standard estimation techniques (as introduced above). However, this world model and observation uncertainty model is tied to one given task and is only in effect during task execution. In contrast, in the proposed framework, the logical model, uncertainties and observations can be defined globally and independently. This allows sensor data processing between the execution of different tasks and the derivation of uncertainty and measurement models for a given task from the global model.

6 Experimental Results

The concepts introduced in this paper have been evaluated in multiple scenarios, one of which uses two cooperating youBots.

6.1 Passing a Baton between two youBots

On the hardware side, each youBot is equipped with a bionic soft gripper [12] mounted at the arm, a WiFi adapter, as well as with retro-reflective markers to provide position tracking. On the system level, both youBots are controlled as separate systems without implicit data transfer, based on a C++ implementation of the Robot Control Core [13] running the onboard computer under Linux with Xenomai extensions to achieve real-time control. However, they are allowed to receive Vicon tracking data through WiFi (that sometimes exhibits



(a) Situation before handover (b) Situation during handover (c) Situation after handover

(d) Model before handover (e) Model during handover (f) Model after handover

Fig. 11. Passing the baton – model and reality

time delays), as well as tasks from an application. On the application level, a single application is used, written in Java based on the Robotics API [14], executed on a standalone computer. However, this application could as well have been executed on any of the youBots.

In this application, a common world model is defined, consisting of the logical model as well as the available sensors and their observations as detailed in the previous sections. For picking up the baton, the raw distance readings of the laser scanner are first filtered, limiting them to distance readings that are within a radius of 50cm around the expected pick up position. Then a pole detection algorithm is used that searches for clusters of a length corresponding to the expected diameter of the baton. The detected cluster position is used as *Sensor Data* for the *Observation* of the *Baton*. Based on the (dynamically estimated) position of the baton, the pickup process is executed. For passing the baton, the application commands the youBot platforms to pass each other, while the gripper is commanded to move towards the center position between the two youBots. Once the distance between the youBots is sufficiently small, the receiving youBot closes its gripper to grasp, followed by the other youBot releasing the baton. When both youBots reach a certain distance after passing each other, the arm motion is stopped, completing the task. These tasks are specified on the full world model, while for execution for each system the tasks are translated into a representation that only uses available data. This way, for each system a corresponding view is automatically derived at runtime and used for command execution, using only the sensor data and estimations available at each youBot.

Fig. 11 shows pictures of an example run[1] of the handover procedure, along with the logical relations and observations present at the corresponding times. During execution, the available estimated positions differ between both youBots: while each youBot knows its start position, it does not know the start position of the other youBot (cf. Fig. 6 and 7). Of course, both youBots could have resorted to a model where the Vicon sensor measurement is used to calculate the position of the youBot relative to the *Origin* (cf. Fig. 7). However, using a more fine-grained model where available (Fig. 6) provides better performance when the tracking system fails: in this case, no further information about motion is available through the Vicon system, but the motion of the youBot relative to its start position can still be tracked using odometry, assuming that the start position remains stationary relative to the *Origin*. This estimation is better than a simple extrapolation of the previous motion based on a constant velocity or acceleration scheme (which has to be applied in the case of Fig. 7), because it takes into account the measured wheel revolutions, and can thus handle non-uniform motions.

To show the utility of this approach and the possible re-use, the same application was used in another system set-up: there, both youBots were configured to be in a single (simulation) system. In this case and without modifying the application or any estimation configuration, at run time only a single world view was created for the simulation system, including estimation models for both youBots similar to Fig. 6. Also in this scenario, the baton could be handed over.

6.2 Tracking a youBot using Distance Estimations

Additionally, in a second scenario, the use of more complex estimation based on an Unscented Kalman Filter was evaluated based on a simulation experiment. Here, a situation similar to

1. A video is available at http://video.isse.de/consistentworldmodel

Fig. 12. Estimation position and ground truth using the generated Kalman filter.

the one depicted in Fig. 10 was modeled. Reusing from the previous example, the youBot *Start Position* was connected to the *Origin* using a *Placement*, and the *Dynamic Connection* between the *Start Position* and the *youBot* was observed through odometry sensors. Additionally, four distance observations of a youBot were used instead of the full pose estimation provided by the Vicon tracking system. For the distance observations, frames at the coordinates $(-3,-3)$, $(-3,10)$, $(10,-3)$, and $(10,10)$ were defined, and the youBot was started at the coordinates $(-2,7)$. The application was executed using the Java Robot Control Core integrated into the Robotics API [14], [15]. The simulation was set up in a way similar to the aforementioned experiments, including simulated Vicon tracking that provides a coordinate frame for the real robot position. However, differing from the previous experiments this sensor data was not modeled as a pose observation about the youBot; instead it was used to calculate the ground truth distances to be provided by the sensors for the modeled distance observations. Additionally, the frame served as an information basis for the orientation observation simulating a compass for the youBot. The simulated youBot was commanded to perform a linear motion in positive X and negative Y direction.

In this situation, the estimators introduced in Sect. 4.1 and 4.2 fail because they cannot handle incomplete observations, e.g. ones that provide only a distance or orientation. Instead, a non-linear filter based Estimator was used that takes the defined logical relations and observations and automatically configures an Unscented Kalman Filter to estimate the positions (or rather degrees of freedom) of the unknown logical relations. Fig. 12 shows an example of the resulting position estimation provided by the automatically configured Kalman filter. Here, estimated as well as ground truth values were sampled at 50 Hz, so the plots show the estimation convergence during the first two seconds of the application run time, showing the ground truth of the youBot as an accelerated motion as well as the estimated position converging towards the correct position.

7 Conclusion

In this paper, we introduced a separation between logical, geometric and measurement information for robotic world models. Logical relationships describe correlations of physical objects via their spatial features which can be reused in different use cases or even applications. Moreover, such correlations are often an inherent part of the model of a robotic device that thus only have to be modeled once. Using the introduced concepts of *Observation*s as measurement definition based on spatial features and *Estimator*s to dynamically integrate the sensor data as geometric information at run-time can be seen as a powerful modeling tool for robot programming, allowing the specification of relationships that can be used by different estimation techniques. Using Estimators that are based on non-linear filters such as Unscented Kalman Filters, complex scenarios with unreliable or partial sensor data can be handled, exploiting the modeled knowledge about the degrees of freedom available in the system as well as the known uncertainty of previous estimations. An important contribution of this paper is that this separation automatically helps to create consistent world views when dealing with distributed robot systems or changing environments, because changes to the logical model are reflected in all World views. Additionally, it allows to automatically configure the estimation process for the given situation using all available sensor data.

Apart from continuous computations at run-time, the modeled relationships can additionally be used for offline-processing, allowing parameter estimation based on recorded sensor values through non-linear optimization. Possible use cases here are to determine the exact position of a tracking marker relative to the robot it is attached to, or to calibrate a robot by performing certain motions, recording the sensor data and optimizing the system parameters in a way to minimize the difference between observations and system model.

References

[1] A. Schierl, A. Angerer, A. Hoffmann, and W. Reif, "Consistent world models for cooperating robots: Separating logical relationships, sensor interpretation and estimation," in *2017 First IEEE International Conference on Robotic Computing (IRC)*, April 2017, pp. 101–108. [Online]. Available: http://ieeexplore.ieee.org/document/7926523/ 1

[2] A. Schierl, A. Angerer, A. Hoffmann, M. Vistein, and W. Reif, "On structure and distribution of software for mobile manipulators," in *Informatics in Control, Automation and Robotics; 12th International Conference, ICINCO 2015; Colmar, France, July 2123, 2015; Revised Selected Papers*, ser. LNEE, J. Filipe, K. Madani, O. Gusikhin, and J. Sasiadek, Eds. Springer, 2016, vol. 383, pp. 209–228. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-31898-1_12 2

[3] E. A. Wan and R. V. D. Merwe, "The unscented kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, 2000, pp. 153–158. [Online]. Available: http://ieeexplore.ieee.org/document/882463/ 4.3

[4] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering*, vol. 82, no. 1, pp. 35–45, 1960. [Online]. Available: http://fluidsengineering.asmedigitalcollection.asme.org/article.aspx?articleid=1430402 5

[5] S. Julier, J. Uhlmann, and H. Durrant-Whyte, "A new approach for filtering nonlinear systems," in *Proceedings of the 1995 American Control Conference*, vol. 3, Jun 1995, pp. 1628–1632. [Online]. Available: http://ieeexplore.ieee.org/document/529783/ 5

[6] B. Arulampalam, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, 2004. [Online]. Available: http://uk.artechhouse.com/-P1308.aspx 5

[7] T. Foote, "tf: The transform library," in *Proceedings of the 2013 IEEE International Conference on Technologies for Practical Robot Applications (TePRA 2013)*, Apr 2013, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6556373/ 5

[8] W. Meeussen. (2010, Oct) Coordinate frames for mobile platforms. Online, accessed Feb 2016. [Online]. Available: http://www.ros.org/reps/rep-0105.html 5

[9] W. Meeussen and D. V. Lu. robot_pose_ekf. Online, accessed Feb 2016. [Online]. Available: http://wiki.ros.org/robot_pose_ekf 5

[10] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, Jul 2014. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-08338-4_25 5

[11] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *The International Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007. [Online]. Available: http://journals.sagepub.com/doi/abs/10.1177/0278364907078091075

[12] BionicTripod with FinGripper – versatile movement and adaptive grasping. Online, accessed Feb 2016. [Online]. Available: https://www.festo.com/cms/de_corp/9779.htm 6.1

[13] M. Vistein, A. Angerer, A. Hoffmann, A. Schierl, and W. Reif, "Flexible and continuous execution of real-time critical robotic tasks," *International Journal of Mechatronics and Automation*, vol. 4, no. 1, pp. 27–38, Jan 2014. [Online]. Available: http://www.inderscienceonline.com/doi/abs/10.1504/IJMA.2014.059773 6.1

[14] A. Angerer, A. Hoffmann, A. Schierl, M. Vistein, and W. Reif, "Robotics API: Object-Oriented Software Development for Industrial Robots," *Journal of Software Engineering for Robotics*, vol. 4, no. 1, pp. 1–22, 2013. [Online]. Available: https://joser.unibg.it/index.php?journal=joser&page=article&op=view&path[]=53 6.1, 6.2

[15] A. Schierl, "Object-oriented modeling and coordination of mobile robots," PhD thesis, Universität Augsburg, 2017. [Online]. Available: https://opus.bibliothek.uni-augsburg.de/opus4/frontdoor/index/index/docId/3915 6.2

Andreas Schierl received his B. Sc. degree in computer science from the University of Augsburg in 2007, the M. Sc. degree in software engineering from the University of Augsburg, the Technische Universitt München and the Ludwig-Maximilians-Universität in 2009 and his doctoral degree in computer science from the University of Augsburg in 2016. Currently, he is a member of the Institute for Software and Systems Engineering, University of Augsburg. He has published about 25 refereed journal and conference papers. His research interest covers mobile robotics, software architectures and software engineering in robotics.

Alwin Hoffmann received his B. Sc. and M. Sc. degrees in information management from the Technische Universitt München and the University of Augsburg, in 2005 and 2007, respectively, and his Diploma and doctoral degree in computer science from the University of Augsburg, in 2008 and 2015. Currently, he is a member of the Institute for Software and Systems Engineering, University of Augsburg. He has published about 35 refereed journals, conference and workshop papers. His research interest covers robotics software engineering, human-robot-collaboration, and multi-robot systems.

Wolfgang Reif is a full professor of Software Engineering, Director of the Institute for Software & Systems Engineering, and Vice-President for Technology and Innovation of the University of Augsburg. He studied Computer Science and received a doctoral degree from the University of Karlsruhe (KIT), Germany. His research interests cover several aspects of Software & Systems Engineering including Safety & Security, Formal Methods, and Self-organizing Systems. A major research focus is on Software in Robotics (collaborative, industrial, assistive, and mobile robotics) with applications in Smart Production (Industry 4.0). He published over 200 scientific papers, articles and book chapters.